Author's Version

High-Performance Elliptical Cone Tracing

U. Emre[1], A. Kanak [1] and S. Steinberg [1]

[1]University of Waterloo, Canada

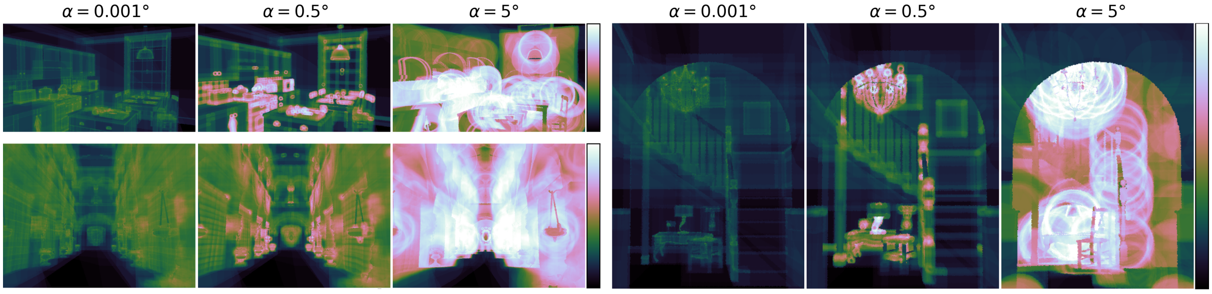


Figure 1: *The performance characteristics of cone traversal change with the cone's size. We visualize the count of internal nodes visited while traversing a BVH with cones of varying aperture as heatmaps (log scale). With thin cones, costs are split between traversal and primitive intersection, but as the cone aperture grows the traversal is dominated by traversing very many internal nodes. Appearance of rings in the heatmaps is attributed to a heuristic we introduce to fast track traversal when a cone contains an entire subtree of a BVH described in Section 4.2. Clockwise from top left scenes are kitchen, staircase and sponza.*

Abstract

In this work, we discuss elliptical cone *traversal in scenes that employ typical triangular meshes. We derive accurate and numerically-stable intersection tests for an elliptical conic frustum with an AABB, plane, edge and a triangle, and analyze the performance of elliptical cone tracing when using different acceleration data structures: SAH-based K-d trees, BVHs as well as a modern 8-wide BVH variant adapted for cone tracing, and compare with ray tracing. In addition, several cone traversal algorithms are analyzed, and we develop novel heuristics and optimizations that give better performance than previous traversal approaches. The results highlight the difference in performance characteristics between rays and cones, and serve to guide the design of acceleration data structures for applications that employ cone tracing.*

CCS Concepts

• ***Computing methodologies*** → *Ray tracing;*

1. Introduction

Varieties of cone tracing have been used in computer graphics for several purposes, for example, global illumination [DBK10, HHK*07], soft shadows [ORM07], acoustics simulations [SRK*09], and furry object rendering [QCH*14]. By providing well-defined non-singular intersection regions, cones allow high-quality anti-aliasing for texture filtering. Cone tracing has also been proposed for wave-optical rendering and RF simulations [SRB*24]. Because cones are inherently able to sample zero-measure features like Dirac delta interactions—e.g., caustics in a unidirectional path tracer or edges for UTD in RF simulations—they provide a more general sampling primitive than rays.

Our motivating application for this work is wave-optical rendering [SP25] where elliptical cones are used as envelopes for wave-optical beams. However, the discussion in this work is kept as general as possible. Contributions in this work are not limited to elliptical cone-tracing but also apply to circular cone-tracing. Our contributions may be useful to a variety of cone-tracing applications such as applications described in [QCH*14, HHK*07, DBK10].

The performance characteristics of a cone tracing workload differ from its ray counterpart (demonstrated in Table 1, Table 2). The costs of intersecting primitives and traversing the internal nodes of an acceleration data structure are considerably different between the workloads. Furthermore, these costs, and their ratio—which is important for SAH-based acceleration data structures—are affected by the cone's shape. This means that acceleration data structures constructed for ray tracing, and ray-tracing focused traversal approaches, are not necessarily optimal for cone tracing: for example, structures that produce a tighter bounding box fit, like bounding volume hierarchies (BVHs), tend to perform considerably better for cone tracing. Furthermore, we also show that some traversal heuristics that are deemed too costly for ray tracing are beneficial for cone tracing.

Several practical challenges and questions in designing high-performance cone tracing algorithms remain insufficiently addressed: First, to our knowledge, no intersection tests for elliptical cones that compute accurate intersection range (nearest and farthest) or intersection points have been published. Boolean tests are insufficient for several applications, where the distance to the intersection and intersection points are needed (for example, sampling edges for UTD for acoustics or RF simulations). Second, analysis of acceleration data structures performance has been limited to simplified cases, e.g., ignoring cone–triangle intersections [WK20], however this ignores the important performance characteristics of different acceleration data structures and traversal heuristics. In addition, modern high-performance BVH implementations often have a higher branching factor—admitting multiple children per node—enabling substantially greater performance through vectorization [FLP*17]. Such wide BVHs have not been adapted to cone tracing.

Our motivation is robust, accurate cone tracing, with arbitrary triangular meshes, including a mix of very small and very large triangles—which may become numerically challenging for cone–primitive intersection tests. The contributions in this paper are:

(a) We derive intersection tests for elliptic conic frusta with AABBs, edges, planes and triangles. Our definition of an elliptic conic frustum (see Section 3) and the intersection tests are designed to remain numerically stable for cones of arbitrary opening half-angles, including very narrow and degenerate cones, which arise in practical applications. In fact, our definition generalizes both elliptical and cylindrical frusta as well as rays.
(b) We analyze the performance of cone tracing with a SAH-based K-d tree and BVH, and analyze a few BVH traversal heuristics.
(c) We adapt an 8-wide BVH for cone tracing, analyze its performance and study a few traversal heuristics.

In this work we target 32-bit IEEE-754 floating points for their performance, and focus on CPU-based workloads; targeting GPUs is left for future work.

2. Related Work

The design of acceleration data structures for cone tracing, and performance analysis of these data structures under cone tracing workloads has received much less attention. Previous work in this area performed such analysis in simplified settings: where leaf traversal is ignored [WK20]; or only K-d trees are considered [ORM07].

Cone–triangle, cone–edge and cone–box intersection tests have been developed for various applications [Hel97,SW10,Ebeb,Ebea]. However, those tests were not designed for the more general case of conic frusta, which are more challenging and they are not numerically stable for small cone opening half-angles or the degenerate case of an elliptical frustum. Further, all are simpler Boolean tests that determine whether an intersection has occurred, but they do not return further information about the geometry of the intersected area. All the above requirements are needed for some applications.

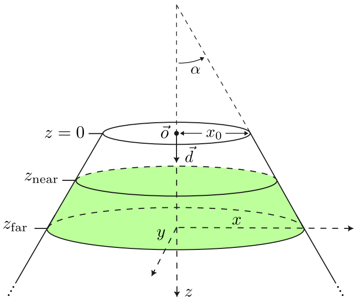


Figure 2: *An elliptic conic frustum (illustrated in green) parametrized by origin* $\vec{\mathbf{o}}$*, directrix* $\vec{\mathbf{d}}$ *(i.e., its mean direction of propagation), initial major axis length* x_0*, and near and far clip planes* z_{near} *and* z_{far}*. The cone's* local frame *is defined such that* x, y *align with the major and minor axes of the elliptical cross section, +z aligns with* $\vec{\mathbf{d}}$*, and the origin is at* $\vec{\mathbf{o}}$*.*

Bounding volume hierarchies (BVHs) with a branching factor higher than 2 [WBB08] are often used today. Such *wide BVHs* enable fast vectorized traversal [FLP*17], and optionally make use of compressed nodes [YKL17, BWWA18] for further acceleration. Previous work in this area has focused on ray traversal; we extend such wide BVHs to cone tracing.

Cone tracing was also discussed as a more general rendering approach [Ama84], for example for anti-aliasing. *Ray cones* methods are used to estimate the filtering footprint, e.g., for texture filtering or selecting shading level-of-detail [AMCB*21, BCAM21].

3. An Elliptic Conic Frustum

We define and parametrize an *elliptic conic frustum* as follows: Let $\vec{\mathbf{o}}$ denote its *base center point*, $x_0 \geq 0$ its *initial major axis length* (major axis length at the base ellipse), $\vec{\mathbf{d}}$ denotes the *cone directrix*, i.e. direction of propagation, $\alpha \geq 0$ is the *half-angle* of the cone, and $\epsilon \in [0,1)$ is the elliptical cone's *eccentricity*, with $\epsilon = 0$ parameterizing an isotropic cone and $\epsilon = 1$ a degenerate flat conic frustum, where the minor axis length is always zero. For convenience we also define the ratio between the major and minor axis lengths as

$$e = \frac{\text{major}}{\text{minor}} = \frac{1}{\sqrt{1-\epsilon^2}} \,. \tag{1}$$

We choose this parameterization because it allows working with elliptical conic frusta with very small half-angles α in a numerically stable manner. This includes the degenerate cases where $\alpha = 0$ (i.e., the cone apex is at ∞). Our definition of an elliptic conic frustum then generalizes elliptical and cylindrical frusta ($\alpha = 0$), as well as rays ($\alpha = x_0 = 0$), both of which do arise in practice in many applications. All of our intersection tests are designed to work with these degenerate cases and remain stable for tiny or vanishing α.

It is often useful to restrict the half-frustum above to a proper frustum via a pair of cutting planes at $z_{\text{near}} \geq 0$ and $z_{\text{far}} \geq 0$, termed the *near* and *far* clip planes, respectively (defined in the cone's local frame, such that $z = 0$ is the origin $\vec{\mathbf{o}}$). See Fig. 2 for an illustration. For example, $\vec{\mathbf{o}}$ can be understood as the cone's sourcing point (e.g., a point on a light source), z_{min} is then chosen to avoid a self intersection with the source, and z_{far} is the (potentially infinite) search distance. Finally, a point $[x, y, z]$, given in local frame, is defined to be inside the elliptical cone if and only if:

$$x^2 + e^2y^2 \le (z\tan\alpha + x_0)^2 \qquad \text{and} \qquad z \in [z_{\text{near}}, z_{\text{far}}] \,. \tag{2}$$

We often work in the cone's local frame, defined such that the x, y, z axes align with the elliptical cone's major axis, minor axis and directrix $\vec{\mathbf{d}}$, respectively, and the origin is at $\vec{\mathbf{o}}$.

3.1. Intersection Tests

We now introduce our intersection tests. These tests compute the closest and farthest intersection points, if intersection occurs. *Intersection distance* is defined as the z distance from $\vec{\mathbf{o}}$ to the point where the cone's flat cross section intersects the primitive, i.e. the distance projected upon the direction $\vec{\mathbf{d}}$, and not the radial distance. All intersection tests take a user-supplied range parameter (`"rng"` in the pseudo-code listings) that clips the cone to $z \in [z_{\text{near}}, z_{\text{far}}]$, as described above.

For brevity the code listings are kept concise; in practice, we use additional early rejection/acceptance queries in some of these tests, which are omitted in paper, please check the accompanying released C++ code. At times, simplified Boolean tests that avoid computing the intersection points are also useful, and can be considerably faster. We provide such Boolean versions of our tests in our accompanying code.

3.1.1. Elliptical Cone–Plane Intersection

The intersection test with a plane returns the intersection distance range over which intersection occurs, as well as the closest and farthest intersection points (the farthest may be at infinity).

Let a plane be parameterized by its normal $\vec{\mathbf{n}}$ and distance from origin d, yielding the plane equation $\vec{\mathbf{p}} \cdot \vec{\mathbf{n}} = d$. Assume that $\vec{\mathbf{n}}, d$ are given in (or are transformed to) the cone's local frame. Let $\vec{\mathbf{p}}$ be a point on the cone's envelope (where the equality in Eq. (2) holds):

$$\vec{\mathbf{p}} = \begin{bmatrix} (z\tan\alpha + x_0)\cos\varphi \\ \frac{1}{e}(z\tan\alpha + x_0)\sin\varphi \\ z \end{bmatrix} \,, \tag{3}$$

where φ is an angle on the xy cross-sectional plane. Plugging the above into the plane equation and solving for z yields:

$$z = \frac{d - x_0\, \vec{\mathbf{n}} \cdot \vec{\mathbf{u}}}{\tan\alpha\, \vec{\mathbf{n}} \cdot \vec{\mathbf{u}} + n_z} \,, \qquad \text{with} \qquad \vec{\mathbf{u}} = \begin{bmatrix} \cos\varphi \\ \frac{1}{e}\sin\varphi \\ 0 \end{bmatrix} \,. \tag{4}$$

We are interested in points $\vec{\mathbf{p}}$ where z is an extremum, therefore we solve for $\frac{\mathrm{d}z}{\mathrm{d}\varphi} = 0$, which yields

$$\vec{\mathbf{u}}_{1,2} = \pm \frac{1}{\sqrt{n_x^2 + e^{-2}n_y^2}} \begin{bmatrix} n_x \\ \frac{1}{e^2}n_y \\ 0 \end{bmatrix} \,. \tag{5}$$

Corresponding $z_{1,2}$ are calculated using Eq. (4).

```
def isect_cone_plane(cone, $\hat{\mathbf{n}}$, $d$, rng):
    transform $\hat{\mathbf{n}}$, $d$ to local cone frame

    # cross sectional intersection position
    $\vec{\mathbf{u}} = \frac{1}{\sqrt{n_x^2+e^{-2}n_y^2}}\left[n_x, \frac{1}{e^2}n_y, 0\right]$
    if isNaN($\vec{\mathbf{u}}$): $\vec{\mathbf{u}}$=0

    $z_{\text{apex}} = -\infty$
    if $x_0 > 0$ and $\alpha > 0$: $z_{\text{apex}} = -\frac{x_0}{\tan\alpha}$

    # intersection candidates
    $z_{1,2} = (d \mp x_0\, \vec{\mathbf{n}} \cdot \vec{\mathbf{u}})/(\pm\tan\alpha\, \vec{\mathbf{n}} \cdot \vec{\mathbf{u}} + n_z)$
    if $z_1 \le z_{apex}$ or isNaN($z_1$): $z_1 = \infty$
    if $z_2 \le z_{apex}$ or isNaN($z_2$): $z_2 = \infty$
    $\vec{\mathbf{p}}_{1,2} = \pm(z_{1,2}\tan\alpha + x_0)\vec{\mathbf{u}} + z_{1,2}\hat{\mathbf{z}}$

    if $z_1 > z_2$:
        swap($z_1$,$z_2$)
        swap($\vec{\mathbf{p}}_1$,$\vec{\mathbf{p}}_2$)

    # clamp intersection points to rng
    if $z_1$ <rng.near:
        $z_1$ = rng.near
        $\vec{\mathbf{p}}_1$ = compute arbitrary point on plane
        ↪  ($\hat{\mathbf{n}}$,$d$) at $z = z_1$ inside the cone
    # (similarly clamp $z_2$,$\vec{\mathbf{p}}_2$ to $z$=rng.far)

    return { .range=Range($z_1$,$z_2$), .nearest=$\vec{\mathbf{p}}_1$,
    ↪  .farthest=$\vec{\mathbf{p}}_2$ }
```

Listing 1: *Elliptical cone–plane intersection test.*

When $n_x^2 + e^{-2}n_y^2$ is tiny or vanishing, care should be taken to avoid NaNs in $\vec{\mathbf{u}}_{1,2}$: in this case we may set $\vec{\mathbf{u}}_{1,2}$ to an arbitrary value as the plane is effectively perpendicular to $\vec{\mathbf{d}}$ and intersects the elliptical cone at a constant z. This is stable: in these cases $\frac{d}{n_z}$ dominates in Eq. (4), and this is indeed the distance to the (almost) perpendicular plane. No intersection occurs when z is $\pm\infty$ or when z is NaN (meaning the plane straddles the cone envelope, but no plane point is strictly inside the cone).

Using the computed $\vec{\mathbf{u}}_{1,2}$ and $z_{1,2}$, we may compute the candidate intersection points $\vec{\mathbf{p}}_{1,2}$. We classify the points, compute the intersection range, and clamp the points to `rng`, depending on the type of conic section that arises on intersection. See Listing 1.

3.1.2. Elliptical Cone–Edge Intersection

The intersection of an elliptical cone and an edge (i.e., a line segment) returns the closest and farthest intersection points, if any. Let the edge be parameterized by two points $\vec{\mathbf{a}}, \vec{\mathbf{b}}$ (assumed to be transformed to the cone's local frame), and we define $\vec{\mathbf{l}} = \vec{\mathbf{b}} - \vec{\mathbf{a}}$. Then, the equation

$$(a_x + tl_x)^2 + e^2(a_y + tl_y)^2 = [(a_z + tl_z)\tan\alpha + x_0]^2 \tag{6}$$

defines a quadratic equation in t for the intersection points, and it is easy to solve for $t_{1,2}$. We use compensated sums to compute the quadratic coefficients and numerically stable expressions for the quadratic roots.

To clamp the intersections to the conic frustum's clip planes, note that if the line $\vec{\mathbf{a}} + t\vec{\mathbf{l}}$ intersects the cone at 2 points, then valid intersection points with the clip planes may only happen between the

```
def isect_cone_edge(cone, $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, rng):
    transform $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$ to local cone frame
    $\vec{\mathbf{l}}$ = $\vec{\mathbf{b}}-\vec{\mathbf{a}}$
    $z_{\text{apex}}$ = $-\infty$
    if $x_0 > 0$ and $\alpha > 0$: $z_{\text{apex}}$ = $-\frac{x_0}{\tan\alpha}$

    $t_{1,2}$ = roots of
    ↪  $(a_x+tl_x)^2+e^2(a_y+tl_y)^2 = [(a_z+tl_z)\tan\alpha+x_0]^2$,
    ↪  otherwise $\infty$

    # discard candidates $t_{1,2}$ behind cone
    if $a_z+t_1l_z < z_{\text{apex}}$: $t_1$ = $\infty$
    if $a_z+t_2l_z \leq z_{\text{apex}}$: $t_2$ = $\infty$
    if $t_1l_z > t_2l_z$: swap($t_1$,$t_2$)

    # clamp to rng
    if $a_z+t_2l_z$<rng.near or $a_z+t_1l_z$>rng.far or
    ↪  $t_1$=$\infty$:
        return $\emptyset$
    if rng.near>$z_{\text{apex}}$ and $a_z+t_1l_z$<rng.near:
        $t_1$ = intersect $\vec{\mathbf{a}}+t\vec{\mathbf{t}}$ with $z$=rng.near
    if $a_z+t_2l_z$>rng.far:
        $t_2$ = intersect $\vec{\mathbf{a}}+t\vec{\mathbf{t}}$ with $z$=rng.far

    $\vec{\mathbf{p}}_{1,2}$ = $\vec{\mathbf{a}}+t_{1,2}\vec{\mathbf{l}}$ if $t_{1,2} \in [0,1]$, otherwise $\infty$
    if $(\vec{\mathbf{p}}_1)_z > (\vec{\mathbf{p}}_2)_z$: swap($\vec{\mathbf{p}}_1$,$\vec{\mathbf{p}}_2$)
    return { .range=Range($(\vec{\mathbf{p}}_1)_z,(\vec{\mathbf{p}}_2)_z$), .nearest=$\vec{\mathbf{p}}_1$,
    ↪  .farthest=$\vec{\mathbf{p}}_2$ }
```

Listing 2: *Elliptical cone–edge intersection test.*

2 line intersections with the cone. If the line intersects the cone at a single point (implying that second point is at $+\infty$), then the edge might intersect the clip planes only after this point. This is expressed succinctly in Listing 2. This test is trivially extended to a line as well.

3.1.3. Elliptical Cone–AABB Intersection

This intersection test is used for acceleration data structure traversal, and returns the intersection distance range over which the elliptical cone intersects the AABB. Either the closest or farthest intersection point must be one of: an AABB vertex; the intersection of the elliptical conic frustum with an AABB's edge; or the intersection of the elliptical conic frustum with a face. See Listing 3 for a pseudocode.

A few optimizations are made: An early rejection can be made if the projected distances of all AABB vertices do not overlap the conic frustum (line 10 in Listing 3). If the closest and farthest AABB vertices are contained in the conic frustum, then we may skip the edge and face tests (lines 16-18). Edges whose vertices are both inside the cone do not need to be tested (line 22). Testing if a cone–plane intersection point is contained in the AABB is done by transforming the point back to world space, and checking if its coordinates that are orthogonal to the face's normal are in the AABB (this is done as we do not need to check the coordinate along the normal and keeps the test stable). We also AVX2 vectorize the transformation and cone–vertex tests (lines 2-18). We found that manual vectorization yields a significant performance improvement compared with the compiler generated code (gcc 14).

```
 1 def isect_cone_aabb(cone, aabb, rng):
 2     transform aabb to local cone frame
 3
 4     $V$ = aabb.verts      # AABB vertices
 5     # AABB vertices that are inside the cone
 6     $C$ = {$\vec{\mathbf{v}}$∈cone | $\vec{\mathbf{v}} \in V$}
 7
 8     possible = rng ∩ Range($\min_{v\in V} v_z$, $\max_{v\in V} v_z$)
 9     # fast reject
10     if possible = $\emptyset$: return $\emptyset$
11
12     ret = Range($\bigcup_{\vec{\mathbf{v}}\in C} v_z$) # range that contains
       ↪  the points $v_z$
13     if $\vec{\mathbf{o}}$ ∈ aabb:
14         ret ∪= Range(0,0) # add origin to the
           ↪  range
15     # fast accept
16     if (($\operatorname{argmin}_{v\in V} v_z$) ∈ cone or $\vec{\mathbf{o}}$ ∈ aabb)
17         and ($\operatorname{argmax}_{v\in V} v_z$) ∈ cone:
18         return ret
19
20     # test AABB edges
21     for $(\vec{\mathbf{a}},\vec{\mathbf{b}})$ in aabb.edges:
22         if $\vec{\mathbf{a}} \notin C$ or $\vec{\mathbf{b}} \notin C$
23             ret ∪= Range(isect_cone_edge(cone,
               ↪  $\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$))
24
25     # test AABB faces
26     for face in aabb.faces:
27         pintr = isect_cone_plane(cone, face,
           ↪  rng)
28         if pintr.nearest∈face:
29             ret.add(pintr.nearest)
30         if pintr.farthest∈face:
31             ret.add(pintr.farthest)
32
33     return ret ∩ possible
```

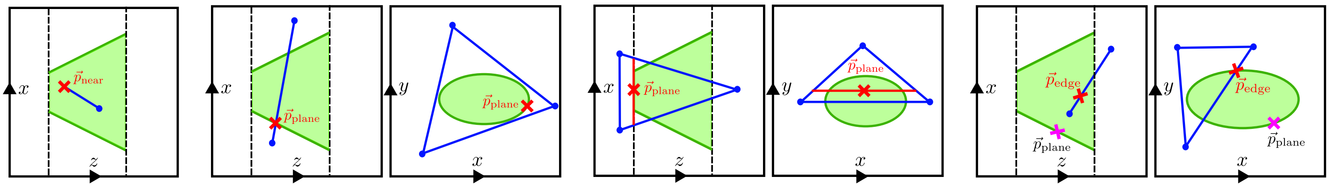
Listing 3: *Elliptical cone–AABB intersection test.*

For many of our results, we only need a simpler Boolean cone–AABB intersection test for traversal. We make the Boolean test available in our released code. To accelerate the Boolean test, a conservative approximation is made: the expensive cone–face tests can be avoided by growing the tested range by the AABB's extent projected upon the cone directrix. Care needs to be taken to avoid growing the range past the cone's apex point. Although this approximation sometimes indicates that there is an intersection when the exact test does not, we show in Section 4.1 that for BVH traversal, the benefits from more efficient Boolean tests outweighs the slight increase in node traversal from the approximation. Other minor optimizations are also made for the simplified Boolean test, see our accompanying code.

3.1.4. Elliptical Cone–Triangle Intersection

This intersection test returns the closest and farthest intersection points, if any, and is summarized in Listing 4. Several cases—illustrated in Fig. 3—are considered in order such that the cheaper cases are evaluated first:

1. **Case 1**: the triangle's nearest or farthest vertex is inside the elliptical conic frustum.
2. **Case 2**: the triangle contains the nearest or farthest cone–plane intersection point. This intersection point can be on the cone's cross section (case 2a) or a clip plane (case 2b).


$\vec{p}_{\mathrm{near}}$
$\vec{p}_{\mathrm{plane}}$
$\vec{p}_{\mathrm{edge}}$
x
y
z

Case 1. nearest vertex is in cone.

Case 2a. cone–plane intersection is on the cross section and inside the triangle.

Case 2b. cone–plane intersection at the near plane and inside the triangle.

Case 3. otherwise, nearest intersection, if any, must be an edge intersection.

Figure 3: *Illustrations of intersection configurations for an elliptical conic frustum with a triangle. The conic frustum is illustrated in green, triangle in blue, and the near and far clip planes using dashed black lines. Red crosses mark the nearest intersection points. The farthest intersection point is handled similarly.*

```
def isect_cone_tri(cone, $\vec{a}$, $\vec{b}$, $\vec{c}$, $\hat{n}$, rng):
    transform $\vec{a}$,$\vec{b}$,$\vec{c}$,$\hat{n}$ to local cone frame
    if all_points_out_of_range($\vec{a}$, $\vec{b}$, $\vec{c}$, rng):
        return None

    # case 1: closest/farthest vertices is in
    ↪   cone
    $\vec{p}_{\mathrm{near}}$ = $\mathrm{argmin}_{v \in V} v_z$
    $\vec{p}_{\mathrm{far}}$ = $\mathrm{argmax}_{v \in V} v_z$
    has_near = $\vec{p}_{\mathrm{near}}$ ∈cone
    has_far = $\vec{p}_{\mathrm{near}}$ ∈cone
    if has_near and has_far:
        return { .nearest=$\vec{p}_{\mathrm{near}}$, .farthest=$\vec{p}_{\mathrm{far}}$ }

    # case 2: cone-plane intersection
    pintr = isect_cone_plane(cone, $\hat{n}$, $\hat{n}\cdot\vec{a}$, rng)
    if not has_near and pintr.nearest∈triangle:
        $\vec{p}_{\mathrm{near}}$ = pintr.nearest
        has_near = true
    if not has_far and pintr.farthest∈triangle:
        $\vec{p}_{\mathrm{far}}$ = pintr.farthest
        has_far = true
    if has_near and has_far:
        return { .nearest=$\vec{p}_{\mathrm{near}}$, .farthest=$\vec{p}_{\mathrm{far}}$ }

    # case 3: intersects edge or no
    ↪   intersection exists
    if not has_near: $\vec{p}_{\mathrm{near}}$ = $\infty$
    if not has_far:  $\vec{p}_{\mathrm{far}}$ = $-\infty$
    for each triangle edge ($\vec{u}$,$\vec{v}$):
        if $\vec{u}$ ∈cone and $\vec{v}$ ∈cone: continue
        eintr = isect_cone_edge(cone, $\vec{u}$, $\vec{v}$)
        if $(\vec{p}_{\mathrm{near}})_z$>Range(eintr).min
            $\vec{p}_{\mathrm{near}}$ = eintr.nearest
        if $(\vec{p}_{\mathrm{far}})_z$<Range(eintr).max
            $\vec{p}_{\mathrm{farthest}}$ = eintr.farthest

    return { .nearest=$\vec{p}_{\mathrm{near}}$, .farthest=$\vec{p}_{\mathrm{far}}$ }
```

Listing 4: *Elliptical cone–triangle intersection test.*

3. **Case 3**: otherwise, nearest or farthest intersection point must be the (nearest or farthest) cone-edge intersection point, if any; otherwise no intersection with the triangle exists.

The numeric stability of this test depends on the numeric stability of the invoked cone–plane and cone–edge intersections tests, as well as the point-in-triangle test. Consider the case where a narrow cone (very small half-angle α) intersects a shared edge between a pair of adjacent triangles, with the edge being very much larger than the cone's cross section. In this case the cone–edge test (case 3) might incorrectly fail due to insufficient precision. However, the cone–plane intersection points for both triangles would essentially be the same points. Therefore, as long as the point-in-triangle test is watertight, at least one of the triangles will be correctly detected as being intersected.

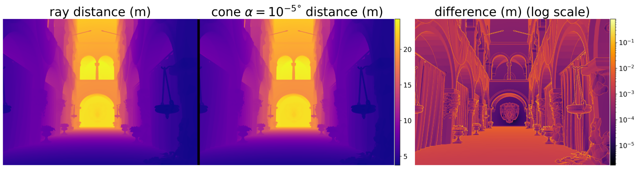

ray distance (m)
cone $\alpha = 10^{-5^\circ}$ distance (m)
difference (m) (log scale)
20
15
10
5
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}

Figure 4: *We test the numerical stability and robustness of cone–triangle intersection by visualizing the distance to closest intersection point for tiny cones with $\alpha = 10^{-5°}$ compared with rays. Cones always intersect the surface at a distance that is equal or less compared with a corresponding ray, though the difference is very small except at object silhouettes.*

We tested cones with tiny half-angles by numerically checking that a cone traversal returns a closer nearest intersection distance compared with ray traversal, for random cones in several scenes; see the visualization in Fig. 4.

4. Acceleration Data Structures

We analyze the performance of tracing elliptical cones with several acceleration data structures (ADS), and discuss several different traversal heuristics. We do not focus on any particular application, but aim to provide a general analysis of the performance of elliptical cones given different ADSs and traversal configurations. Real applications will likely to want to perform additional steps: for example, for the rendering soft shadows the relative cross-sectional area that is intersected by primitives would be computed, while for RF simulations intersected edges would be classified. Our analyzed ADSs are: a typical SAH-based K-d tree [WH06] and a BVH [GS87]. We also adapt a modern, vectorized 8-wide BVH for cone tracing. All our tests were run on an AMD Ryzen™ Threadripper™ PRO 5975WX.

Performance metrics for several scenes are given in Table 1 for primary rays and primary cones with a variety of cone half-angles α. The time per primary ray and primary cones of varying half-angles is visualized in Fig. 7, highlighting that smaller cones ($< 5^\circ$ cone α) favour our implementation of the 8-wide BVH, while larger

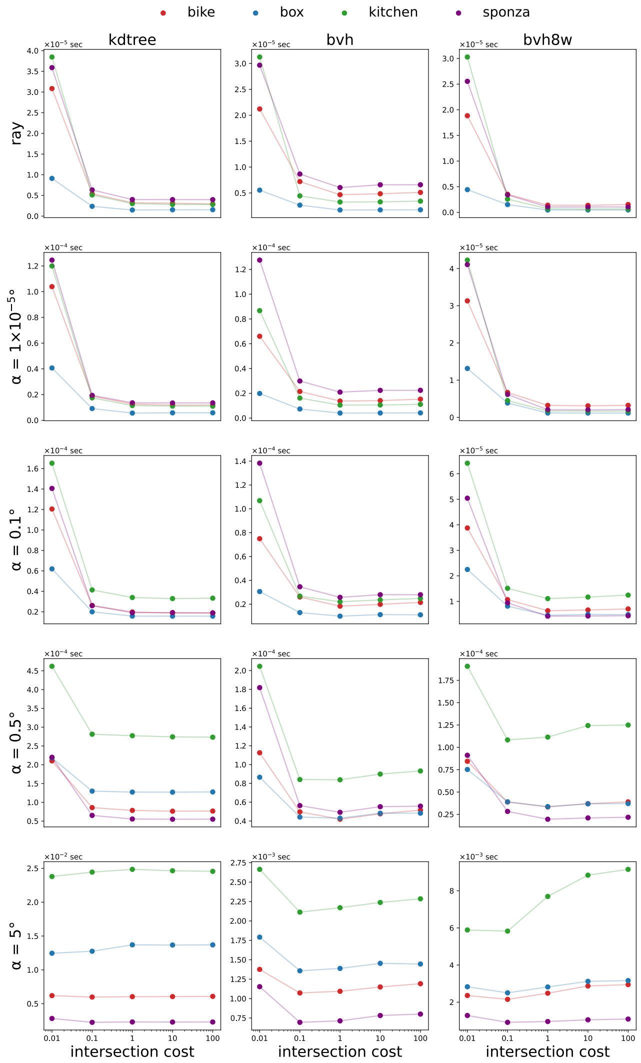


Figure 5: *Average time to trace a cone as a function of the primitive intersection-to-traversal cost ratio parameter. Larger cone apertures perform better with shallower ADSs.*

cones ($\geq 5^\circ$ cone α) favour our implementation of the BVH. The performance of secondary cones of random eccentricity is also explored in Table 2. Additional figures analyzing performance characteristics of secondary cones is included in our supplemental document. The conclusions regarding the performance of the traversal heuristics discussed in this paper and the differences between the ADSs explored in this paper remain the same for secondary cones.

Construction SAH-based ADSs construction makes use of traversal cost and primitive intersection cost constants during construction. In Fig. 5 we compare traversal performance with different ratios of these cost constants. For ray tracing, a traversal-to-primitive-intersection cost ratio of roughly 1 is usually optimal, and for small cones this remains a performant choice, enabling mixed workloads (ray and cone tracing) with a single ADS—useful for many applications. However, for larger cones this ratio becomes suboptimal: larger cones benefit from shallower ADSs, as they traverse a greater part of the tree. Deciding on an optimal cost ratio depends on the targeted workload. We used a traversal-to-primitive-intersection cost ratio of 1 in all our results.

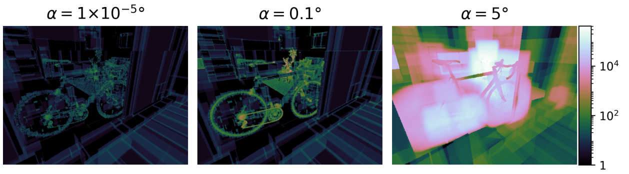


Figure 6: *Heatmaps (log scale) of revisited triangle count in a K-d tree traversal. The count of revisited triangles increases with cone angle similar to the count of intersected triangles. Revisited triangles constitute a considerable wasted effort in K-d tree traversal.*

Traversal Under a CPU ray-tracing workload K-d trees often outperform binary BVHs. For small cones this trend continues to hold. Finding the intersection of a ray with the splitting plane is very cheap, driving fast ray traversal. Similarly, intersecting a cone with the splitting plane is a cheaper traversal step compared to the intersections of a cone with AABBs when traversing a BVH.

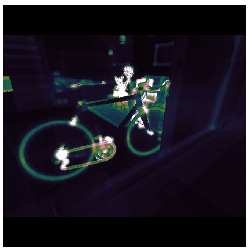
For larger cones we see BVHs significantly outperform K-d trees which can be attributed to the following: (i) a cone may intersect multiple primitives, a dynamically-allocated data structure (or mailboxing [APB87, AW*87]) is used to keep track of which triangles we have encountered, saving on repeated very expensive cone–triangle intersection tests (see Fig. 6). This is not needed for a BVH, as a BVH is an object partitioning scheme. And most importantly, (ii) BVHs provide a tighter bounding box fit for the leaf nodes, reducing the number of tested triangles. This result contradicts Wiche et al. [WK20]: because they ignore the expensive cone–triangle intersection costs and only consider traversal.

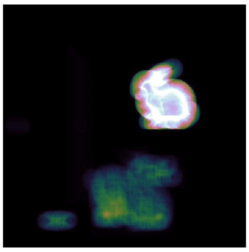
Shadow Queries We also test cone shadow queries (Boolean tests that stop as soon as a cone–triangle hit is found along the search range), see Fig. 8. The usefulness of such queries is highly application dependent. As expected, with larger cones the shadow queries are relatively much faster. This is because full queries become more expensive as cone size grows while partial intersections can be found earlier in large cones. The 8-wide BVH and K-d tree see the greatest speedup for larger cones as they traverse internal nodes faster.

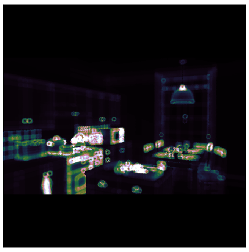
4.1. BVH Traversal Heuristics

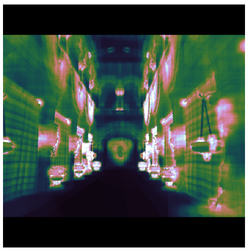
When traversing BVH nodes with a cone, the order of traversing the child nodes of an interior node can be determined in several ways:

1. By computing the exact nearest cone–AABB intersection point using the full, expensive test (Section 3.1.3).
2. Instead of using the full test, the simpler and more efficient Boolean test introduced in Section 3.1.3 can be used for traversal, while child ordering is done via one of the following approximative distances to the AABB:









			bike			box			kitchen			sponza		
		cone α	µs	nodes	tris	µs	nodes	tris	µs	nodes	tris	µs	nodes	tris
primary cones	kdtree	ray	3.22	44.0	25.3	1.48	15.4	10.6	2.98	34.0	24.8	3.99	51.5	31.3
		10^{-5}°	12.5	47.2	31.0	5.81	19.5	11.9	11.5	39.4	29.4	13.5	54.1	34.2
		0.1°	19.5	50.2	47.5	15.8	28.9	38.2	33.9	50.3	88.4	19.2	57.9	46.7
		0.5°	77.8	70.9	198	129	122	349	276	152	795	55.1	80.1	129
		5°	6.04K	1.36K	12.1K	13.7K	6.12K	22.6K	24.7K	6.14K	46.7K	2.31K	1.06K	4.51K
		15°	80.8K	11.6K	109K	57.2K	21.9K	81.7K	243K	46.0K	354K	20.9K	6.94K	31.3K
	bvh	ray	4.66	25.2	11.7	1.72	9.17	2.48	3.32	20.4	3.58	5.95	39.0	3.95
		10^{-5}°	13.6	33.5	18.4	4.15	11.0	3.24	10.5	29.5	10.6	20.7	63.1	16.8
		0.1°	18.3	37.4	25.2	10.1	20.1	12.7	22.0	43.2	28.3	25.7	68.0	21.7
		0.5°	42.1	59.6	62.6	42.1	71.3	59.7	84.1	114	113	49.0	95.1	51.9
		5°	1.08K	445	628	1.38K	637	543	2.18K	956	1.07K	714	650	636
		15°	13.4K	1.67K	2.23K	5.55K	316	277	17.7K	2.86K	3.26K	4.12K	1.94K	1.96K
	bvh8w	ray	1.37	10.0	11.1	.477	3.37	2.49	.701	7.61	3.58	1.04	15.6	5.10
		10^{-5}°	3.16	9.06	13.5	1.22	3.91	3.24	1.69	7.80	5.26	2.05	13.8	5.35
		0.1°	6.45	11.0	24.0	4.49	7.45	14.7	11.2	15.1	38.2	4.24	15.9	11.5
		0.5°	33.4	27.7	126	33.3	34.7	118	112	91.8	456	19.9	29.4	61.1
		5°	2.49K	1.30K	8.52K	2.81K	1.59K	7.90K	7.63K	5.72K	34.9K	945	632	2.99K
		15°	26.4K	11.2K	72.7K	4.47K	738	3.85K	62.0K	41.0K	258K	7.13K	4.06K	21.6K

Table 1: *Mean time, nodes visited and triangles intersected per traversed primary cone or ray for different ADSs and several scenes.*

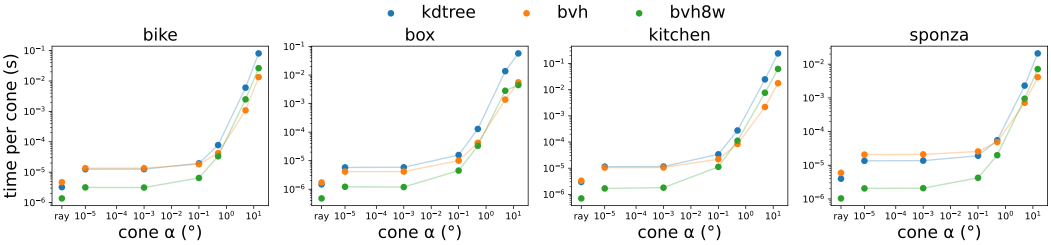


Figure 7: *Mean time per traversed primary cone or ray for different ADSs and several scenes visualized from Table 1. Smaller cones ($< 5°$ cone α) favour our implementation of the 8-wide BVH, while larger cones ($\geq 5°$ cone α) favour our implementation of the BVH.*

a. the ray–AABB intersection distance, where the ray is the cone directrix $\vec{\mathbf{d}}$; or
b. the distance to the AABB midpoint, projected upon the cone directrix.

Measurements of the performance of these three traversal ordering schemes are shown in Fig. 9. As expected, the number of visited BVH nodes and triangles is usually reduced with the exact cone–AABB test (though this is scene dependent), however this is offsetted by significantly more expensive traversal cost. In practice, using one of the approximate distance heuristics almost always makes for a considerably faster traversal. In the rest of our results we use the cone directrix-to-AABB distance heuristic (2a above).

4.2. BVH Contained Subtree Traversal

Another useful heuristic for cone tracing is checking if a BVH's internal node is fully contained within a cone during traversal. This applies both to the binary BVH and the 8-wide BVH. Testing if a bounding box is contained within the cone is done via testing if all 8 box vertices are within the cone using a single vectorized test. Every node stores a pointer to the list of all triangles within the node's subtree. Once we detect that a node (and its subtree) is fully contained in a cone, we ignore child nodes and proceed to compute the intersection ranges with all the subtree triangles. As we know that all the triangles within that subtree are fully contained in the cone as well, expensive full cone–triangle intersection tests are not

			bike			**box**			**kitchen**			**sponza**		
		cone α	µs	nodes	tris	µs	nodes	tris	µs	nodes	tris	µs	nodes	tris
secondary cones	kdtree	ray	2.39	27.6	13.1	1.70	16.4	12.4	2.05	18.6	11.4	3.07	33.5	19.8
		$10^{-5\circ}$	8.27	29.6	15.1	6.05	20.0	11.8	6.67	20.4	12.4	11.0	35.6	23.5
		0.1°	12.2	31.1	23.0	7.00	20.7	13.6	10.9	22.1	20.9	13.0	36.9	27.5
		0.5°	63.8	44.5	125	14.5	25.7	28.4	50.5	34.9	106	26.2	44.5	53.8
		5°	4.27K	735	6.23K	399	222	710	2.72K	659	4.78K	618	314	1.14K
		15°	36.6K	4.97K	44.7K	3.11K	1.35K	4.77K	22.9K	4.58K	34.6K	4.80K	1.79K	7.23K
	bvh	ray	3.33	17.0	4.44	1.67	8.70	2.50	2.36	11.2	2.85	5.57	32.3	3.47
		$10^{-5\circ}$	13.5	26.2	11.1	5.50	10.8	4.65	8.77	16.5	6.64	24.7	53.2	15.7
		0.1°	15.7	28.1	13.8	6.06	11.4	5.20	10.5	18.0	8.58	26.4	54.7	17.3
		0.5°	28.6	39.5	29.3	8.93	14.8	8.82	19.1	26.3	19.1	34.6	62.6	26.0
		5°	568	227	263	58.4	67.4	57.0	228	133	146	195	208	187
		15°	5.70K	776	928	248	180	155	1.52K	425	487	903	604	620
	bvh8w	ray	1.00	6.70	4.72	.478	3.79	2.60	.714	4.18	2.99	1.10	12.7	4.79
		$10^{-5\circ}$	1.92	6.54	4.96	1.25	4.15	3.18	1.52	4.27	3.46	1.94	11.7	4.42
		0.1°	7.37	10.1	24.5	1.82	4.61	4.88	4.10	5.87	11.7	3.23	12.8	7.99
		0.5°	55.2	38.2	196	6.65	8.39	21.5	37.7	27.6	140	14.4	21.8	48.5
		5°	2.64K	1.46K	9.15K	234	162	802	1.92K	1.30K	8.08K	445	340	1.73K
		15°	15.2K	7.00K	44.7K	1.13K	722	3.72K	10.9K	7.65K	47.8K	2.44K	1.76K	9.61K

Table 2: *Mean time, nodes visited and triangles intersected per traversed secondary cone or ray for different ADSs and a couple of scenes. Secondary cones have randomly selected eccentricity: the displayed cone α indicates the total solid angle of the cone.*

needed. Finding the intersection range with each triangle simplifies considerably to finding the closest and farthest triangle vertices, which is also done in an AVX2 vectorized manner.

This proposed subtree traversal solves the *accelerator-in-a-cone* problem highlighted by Wiche et al. [WK20]. Measurements of performance with and without this optimization are given in Fig. 10. We use this optimization for BVHs and 8-wide BVHs in all our results. The effects of subtree traversal are visualized in Fig. 1, where we plot the count of traversed internal nodes as heatmaps: the rings that arise visualize where subtree traversal serves to reduce the count of nodes that need to be traversed.

4.3. 8-Wide BVH

We construct an 8-wide BVH from the binary SAH-based BVH by collapsing every 3 levels into one, in similar manner to Fuetterling et al. [FLP*17]. Nodes in the 8-wide BVH store the full 8 AABBs in a vectorized form, and loading of the nodes and traversal is AVX2-accelerated for fast vectorized traversal. We also experimented with compressing the 8-wide nodes using 8-bit or 16-bit quantization, however even with fast vectorized AVX2 decompression this proves to be a performance loss (suggesting that traversal performance is ALU limited and not memory bound, at least for primary rays). This is likely to change on a modern GPU.

With ray tracing 8-wide BVH traversal is performed via vectorized ray–AABB cluster intersection. Vectorizing the significantly more complex full cone–AABB intersection test is cumbersome, due to branching, and is unlikely to yield a meaningful performance boost. Furthermore, just as with the binary BVH, we prefer to avoid doing the full cone–AABB test, and instead employ a faster heuristic for traversal: We perform a vectorized ray–AABB cluster intersection test that conservatively approximates the cone–AABB cluster test by enlarging the AABBs by the cone's cross-sectional size, see Fig. 12 for an illustration. Pseudocode is given in Listing 5. This works well when the cone half-angle α is not too large.

With the optimization above, the cone traversal performance of the 8-wide BVH is very good. For smaller cones (< 0.5° cone α) the 8-wide BVH spends a much larger fraction of traversal time on traversing leafs and running expensive cone–triangle intersection tests compared to other ADSs, see Fig. 13. However, this fraction decreases following the trend in the degradation of the 8-wide BVH's relative performance for larger cones. We attribute this to the wasted effort caused by false positives arising from our conservative cone–AABB cluster intersection test worsening with larger cones. To minimize the impact of these false positives, with respect to wasted cone–triangle intersections in particular, we tried employing a heuristic where we perform an accurate cone–AABB Boolean intersection test whenever we start traversing a leaf node. As expected, there is a performance increase (varying by scene) for larger cones where many cone–triangle intersections are avoided, see Fig. 11. However, the overhead of a cone–AABB test per leaf contributes to a slowdown for smaller cones. We do not use this heuristic in our results.

The final heuristic addresses child traversal order. In a wide BVH, child nodes are often either visited in a fixed order, e.g. in an octant-based fashion [YKL17], or by sorting based on hit distance. Previous work targeting ray-tracing workloads [YKL17, FLP*17] favored a fixed traversal order as the sorting expense was deemed unfavorable. However, as leaf traversal with cone tracing is expensive, we find that performing a simple insertion sort of the (up to) 8 intersected child nodes based on hit distance (computed via the

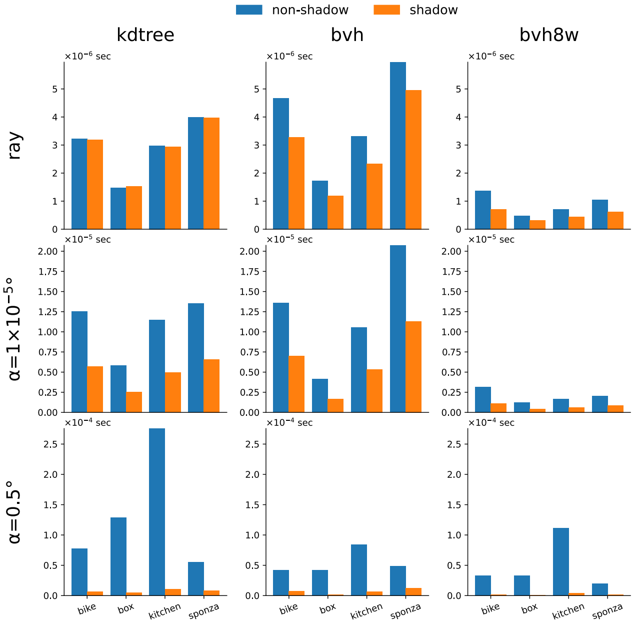


Figure 8: *Shadow cone queries compared with full cone queries. As expected, with greater cone apertures shadow queries get comparatively faster as they can terminate earlier while the full queries become more expensive; and, ADSs that are able to spend less time on traversal (K-d tree, 8-wide BVH) benefit more, compared to ADSs that are more bound by traversal costs (BVH).*

conservative ray–AABB cluster intersection test outlined above) gives a performance benefit.

5. Future Work

Future research could explore cone tracing on the GPU, the use of oriented bounding boxes in acceleration data structures and additional traversal optimizations/heuristics for cones with very large half-opening angles.

6. Conclusion

This work aims to fill the gaps missing in published work, condense knowledge, and analyze best practices regarding practical cone tracing with triangular meshes. We presented intersection tests of elliptical conic frusta with planes, edges, AABBs and triangles. Our supplemental material includes C++ source code for these intersection tests, as well as additional experiments and performance data. We also discussed acceleration data structures, traversal heuristics as well as a wide BVH designed for cone tracing workloads. Our results serve to demonstrate the difference in performance characteristics between cones with different aperture sizes and rays across several ADSs; as well as guide the selection of a suitable ADS for a cone tracing application.

Acknowledgements

We would like to thank Matt Pharr for invaluable discussion, input and comments for this work.

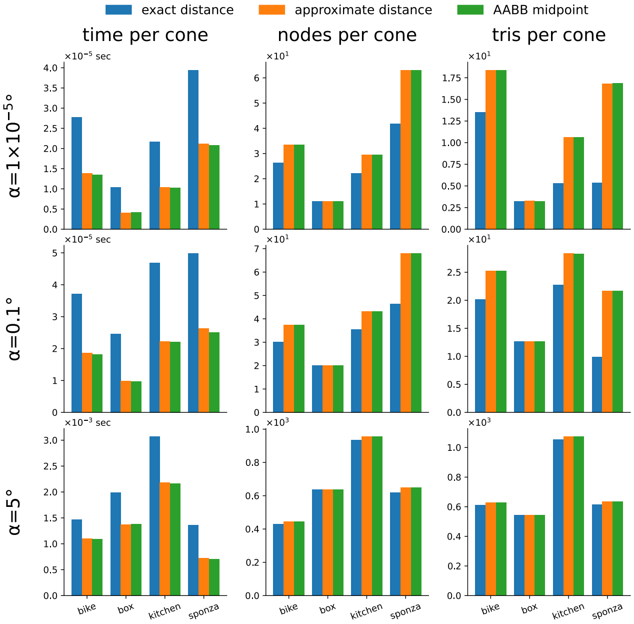


Figure 9: *Average time, count of nodes visited, and triangles tested per cone traversal with a binary BVH. Compared is the exact cone-to-child node distance (blue) against two approximations: ray (cone directrix)-to-node distance (orange), and origin-to-midpoint distance (green). The exact distance heuristic enables traversing less nodes and triangles, but is not worth the significant additional cost of a full cone–AABB intersection test.*

References

[Ama84] Amanatides J.: Ray tracing with cones. *Comput. Graph. (ACM) 18*, 3 (July 1984), 129–135. 2

[AMCB*21] Akenine-Möller T., Crassin C., Boksansky J., Belcour L., Panteleev A., Wright O.: Improved shader and texture level of detail using ray cones. *Journal of Computer Graphics Techniques Vol 10*, 1 (2021), 12–18. 2

[APB87] Arnaldi B., Priol T., Bouatouch K.: A new space subdivision method for ray tracing csg modelled scenes. *The Visual Computer 3* (1987), 98–108. doi:10.1007/BF02153666. 6

[AW*87] Amanatides J., Woo A., et al.: A fast voxel traversal algorithm for ray tracing. In *Eurographics* (1987), vol. 87, pp. 3–10. doi:10.2312/egtp.19871000. 6

[BCAM21] Boksansky J., Crassin C., Akenine-Möller T.: Refraction ray cones for texture level of detail. *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX* (2021), 115–125. 2

[BWWA18] Benthin C., Wald I., Woop S., Áfra A. T.: Compressed-leaf bounding volume hierarchies. In *Proceedings of the Conference on High-Performance Graphics* (New York, NY, USA, 2018), HPG '18, Association for Computing Machinery. doi:10.1145/3231578.3231581. 2

[DBK10] Duvenhage B., Bouatouch K., Kourie D.: Exploring the use of glossy light volumes for interactive global illumination. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa* (New York, NY, USA, 2010), AFRIGRAPH '10, Association for Computing Machinery, p. 139–148. doi:10.1145/1811158.1811181. 1

[Ebea] Eberly D.: Intersection of a box and a cone or cone frustum.

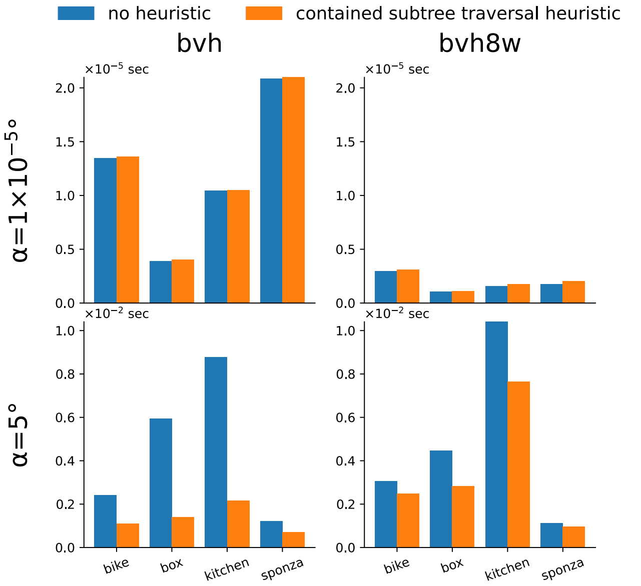


Figure 10: *Average time per cone traversal in a binary BVH and 8-wide BVH. Comparison is made for time spent with (orange) and without (blue) the contained subtree traversal heuristic. Checking if each node is fully contained adds a small overhead that can slow cone traversal down for very tiny cones, but for larger cones where many nodes are contained there is a significant speedup from avoiding intersection tests.*

https://www.geometrictools.com/Documentation/IntersectionBoxCone.pdf. Updated: September 11, 2020. 2

[Ebeb] Eberly D.: Intersection of a triangle and a cone. https://geometrictools.com/Documentation/IntersectionTriangleCone.pdf. Updated: March 2, 2008. 2

[FLP*17] Fuetterling V., Lojewski C., Pfreundt F.-J., Hamann B., Ebert A.: Accelerated single ray tracing for wide vector units. In *Proceedings of High Performance Graphics* (New York, NY, USA, July 2017), ACM. doi:10.1145/3105762.3105785. 2, 8

[GS87] Goldsmith J., Salmon J.: Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (1987), 14–20. doi:10.1109/MCG.1987.276983. 5

[Hel97] Held M.: Erit—a collection of efficient and reliable intersection tests. *Journal of Graphics Tools 2*, 4 (1997), 25–44. doi:10.1080/10867651.1997.10487482. 2

[HHK*07] Herzog R., Havran V., Kinuwaki S., Myszkowski K., Seidel H.-P.: Global Illumination using Photon Ray Splatting . *Computer Graphics Forum* (2007). doi:10.1111/j.1467-8659.2007.01073.x. 1

[ORM07] Overbeck R., Ramamoorthi R., Mark W. R.: A real-time beam tracer with application to exact soft shadows. In *Proceedings of the 18th Eurographics conference on Rendering Techniques* (2007), pp. 85–98. doi:10.2312/EGWR/EGSR07/085-098. 1, 2

[QCH*14] Qin H., Chai M., Hou Q., Ren Z., Zhou K.: Cone tracing for furry object rendering. *IEEE Transactions on Visualization and Computer Graphics 20*, 8 (2014), 1178–1188. doi:10.1109/TVCG.2013.270. 1

[SP25] Steinberg S., Pharr M.: Wave tracing: Generalizing the path integral to wave optics, 2025. URL: https://arxiv.org/abs/2508.17386, arXiv:2508.17386. 1

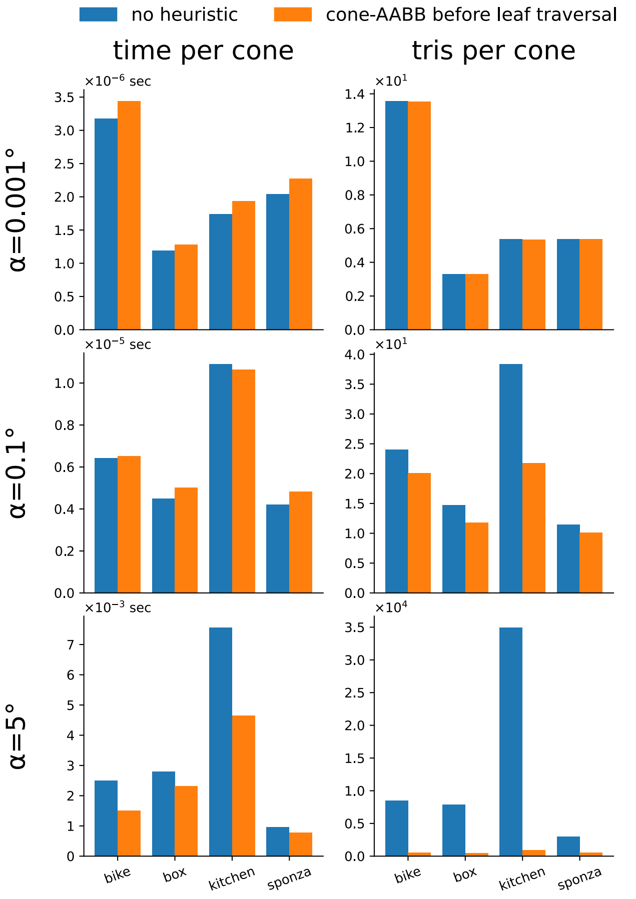


Figure 11: *Comparison of doing a cone-AABB test before traversing leafs in the 8-wide BVH (orange) and without the extra cone-AABB test (blue). For larger cones this heuristic avoids doing a significant number of cone-triangle intersections, which gives a performance benefit as cone size increases. However, the overhead of a cone-AABB test per leaf makes this a performance loss for smaller cones in most cases.*

[SRB*24] Steinberg S., Ramamoorthi R., Bitterli B., d'Eon E., Yan L.-Q., Pharr M.: A generalized ray formulation for wave-optical light transport. *ACM Trans. Graph. 43*, 6 (Nov. 2024). doi:10.1145/3687902. 1

[SRK*09] Schmitz A., Rick T., Karolski T., Kuhlen T. W., Kobbelt L.: Simulation of radio wave propagation by beam tracing. In *EGPGV@ Eurographics* (2009), pp. 17–24. doi:http://dx.doi.org/10.2312/EGPGV/EGPGV09/017-024. 1

[SW10] Sevilla D., Wachsmuth D.: Polynomial integration on regions defined by a triangle and a conic. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation* (New York, NY, USA, July 2010), ACM. doi:10.1145/1837934.1837968. 2

[WBB08] Wald I., Benthin C., Boulos S.: Getting rid of packets - efficient SIMD single-ray traversal using multi-branching BVHs -. In *2008 IEEE Symposium on Interactive Ray Tracing* (Aug. 2008), IEEE. doi:10.1109/RT.2008.4634620. 2

[WH06] Wald I., Havran V.: On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *2006 IEEE Symposium on Interac-*

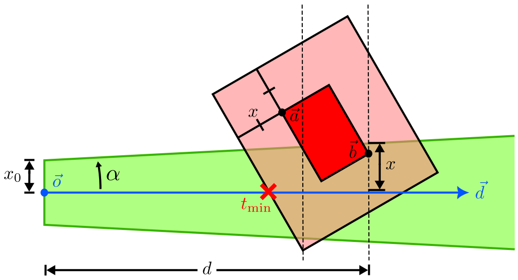


Figure 12: *Elliptical cone–AABB intersection for 8-wide BVH is done via a conservative ray–AABB intersection with the AABB enlarged by the cone's maximal cross section size x. Dashed lines indicate accurate cone–AABB intersection range, while the nearest point (red cross) computed with this test serves a conservative approximation.*

```
def isect_cone_aabb_8w(cone, aabbs, max_dist):
    # o⃗ – cone origin, d⃗ – cone directrix
    invdx = set1_ps(1/dₓ)
    minx = sub_ps(aabbs.minx, set1_ps(oₓ))
    maxx = sub_ps(aabbs.maxx, set1_ps(oₓ))
    ax = blendv_ps(minx, maxx, invdx)
    bx = blendv_ps(minx, maxx, -invdx)
    # repeat all of the above for y and z...

    # d⃗·b⃗ is farthest z from o⃗ of all AABBs
    zdist = fmadd_ps(dx, bx, fmadd_ps(dy, by,
    ↪  mul_ps(dz, bz)))
    # greatest cone cross-section size
    s = fmadd_ps(zdist, set1_ps(tanα),
    ↪  set1_ps(x₀))

    # enlarge AABBs and ray--AABBs cluster
    ↪  intersection
    enlrx = blendv_ps(s, -s, invdx)
    ax = sub_ps(ax, enlrx)
    bx = add_ps(bx, enlrx)
    ux = mul_ps(ax, invdx)
    vx = mul_ps(bx, invdx)
    # repeat all of the above for y and z...

    tmin = max_ps(max_ps(set1_ps(0), ux),
    ↪  max_ps(uy, uz))
    tmax = min_ps(vx, min_ps(vy, vz))
    test = and_ps(cmp_ps(tmin,tmax,LE_OQ),
    ↪  cmp_ps(tmin,set1_ps(max_dist),LT_OQ))
    return { .results = test, .dists = tmin }
```

Listing 5: *Vectorized elliptical cone–AABB 8-wide cluster intersection test. The struct 'aabbs' contains the vectorized coordinates of the AABBs' min and max vertices.*

tive Ray Tracing (Sept. 2006), IEEE, p. 61–69. doi:10.1109/rt.2006.280216. 5

[WK20] WICHE R., KURI D.: Performance evaluation of acceleration structures for cone-tracing traversal. *Journal of Computer Graphics Techniques Vol 9*, 1 (2020). URL: https://jcgt.org/published/0009/01/01/. 2, 6, 8

[YKL17] YLITIE H., KARRAS T., LAINE S.: Efficient incoherent ray traversal on gpus through compressed wide bvhs. In *Proceedings of High Performance Graphics* (New York, NY, USA, 2017), HPG '17, Association for Computing Machinery. doi:10.1145/3105762.3105773. 2, 8

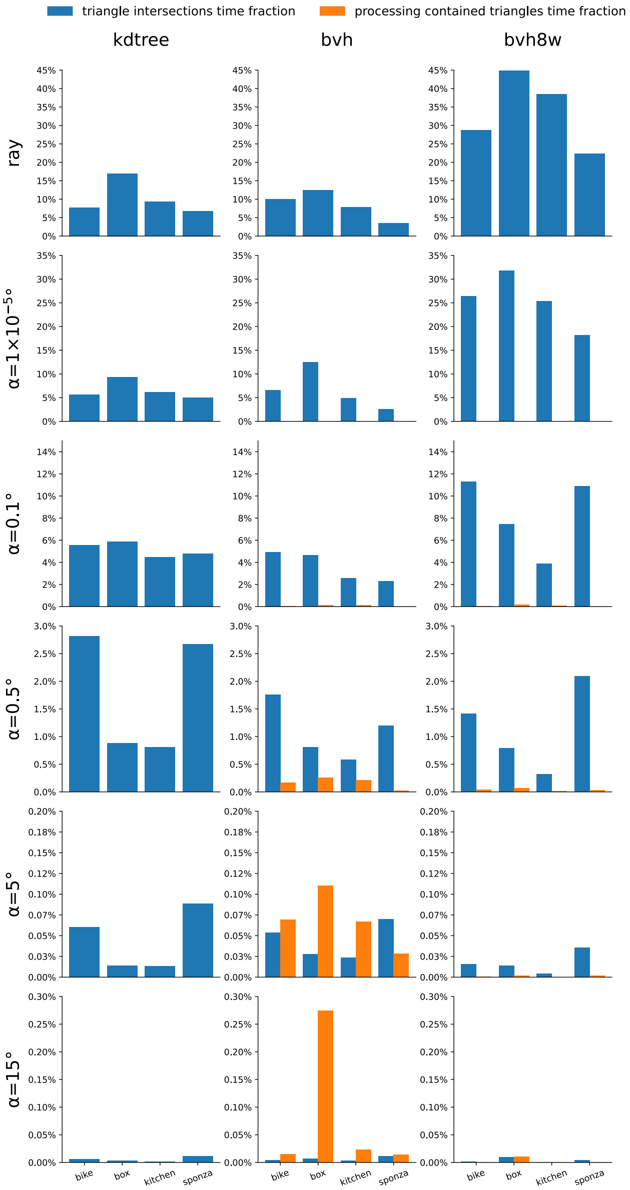


Figure 13: *Relative time spent doing triangle intersections tests or determining intersection range for triangles in the case of contained subtree traversals (as described in Section 4.2) when traversing an ADS. The 8-wide BVH is considerably faster at traversing internal ADS nodes, and often spends less time doing so for rays and small cones. However, for large cones 8-wide BVH traversal visits significantly more nodes as shown in Table 1 which explains why it spends less relative time on triangles during traversal.*